Internal 2.0
- User Guide -

C. Benoit, G. Jeanfaivre, S. Peron, P. Raud, S. Landier, B. Rodriguez, S. Verley
Onera / DSNA

1 Internal: pyTree basic management

1.1 Preamble

This module provides routine for pyTree basic management.

To use the module:

```
import Converter.Internal as Internal
```

All module functions of Converter.PyTree, Transform.PyTree, ... work on 3 types of containers: a container for grid coordinates named __GridCoordinates__, a container for node solution named __FlowSolutionNodes__ and a container for center solution named __FlowSolutionCenters__. By default:

```
Internal.__GridCoordinates__ = 'GridCoordinates'
Internal.__FlowSolutionNodes__ = 'FlowSolution'
Internal.__FlowSolutionCenters__ = 'FlowSolution#Centers'
```

To make the function operate on another named container for example 'FlowSolution#Init', type:

```
Internal.__FlowSolutionNodes__ = 'FlowSolution#Init'
```

1.2 Node tests

Internal.isTopTree: return True if node is a top tree:

```
res = Internal.isTopTree(node)
```

(See: Examples/Converter/isTopTreePT.py)

Internal.isStdNode: return 0 if t is a list of standard nodes, -1 if t is a standard node, -2 otherwise:

```
res = Internal.isStdNode(node)
```

(See: Examples/Converter/isStdNodePT.py)

Internal.typeOfNode: return 1 if node is a zone node, 2 if node is a list of zones, 3 if node is a tree, 4 if node is a base, 5 if node is a list of bases, -1 otherwise:

```
res = Internal.typeOfNode(node)
```

(See: Examples/Converter/typeOfNodePT.py)

Internal.isType: compare given type and node type. Wildcards are accepted:

```
res = Internal.isType(node, 'Zone_t')
```

(See: Examples/Converter/isTypePT.py)

Internal.isName: compare given name and node name. Wildcards are accepted:

```
res = Internal.isName(node, 'Zone*')
```

(See: Examples/Converter/isNamePT.py)

Internal.isValue: compare given value and node value. If value is a string, accepts wildcards, accepts also numpy arrays as value:

```
res = Internal.isValue(node, 1.)
```

(See: Examples/Converter/isValuePT.py)

1.3 Set/Create nodes

Internal.setValue: set the given value in node (node is modified):

```
Internal.setValue(node, 1.)
```

(See: Examples/Converter/setValueIPT.py)

Internal.setName: set the given name in node (node is modified):

```
Internal.setName(node, 'myNode')
```

(See: Examples/Converter/setNamePT.py)

Internal.setType: set the given type in node (node is modified):

```
Internal.setType(node, 'Zone_t')
```

(See: Examples/Converter/setTypePT.py)

Internal.createNode: create a node given name and type and optional value and children:

```
node = Internal.createNode('myNode', 'DataArray_t', value=1., children=[])
```

(See: Examples/Converter/createNodePT.py)

Internal.addChild: add a child node to a given node. Position in children list can be specified (node is modified):

```
node = Internal.addChild(node, child, pos=-1)
```

(See: Examples/Converter/addChildPT.py)

Internal.createChild: create a child node and attach it to a given node. Child's node name, type, value and children can be specified. Position in children list can also be specified (node is modified):

```
node = Internal.createChild(node, 'myNode', 'DataArray_t', value=None, children=[], pos=-1)
```

(See: Examples/Converter/createChildPT.py)

Internal.createUniqueChild: same as createChild except that, if a node with name already exists in the children list, it is replaced by the new values (node is modified):

```
node = Internal.createUniqueChild(node, 'myNode', 'DataArray_t', value=None)
```

(See: Examples/Converter/createUniqueChildPT.py)

1.4 Access

Internal.getNodeFromPath: return a node from its path string. Note that node path is relative to input node A:

```
node = Internal.getNodeFromPath(A, "Base/Zone")
```

(See: Examples/Converter/getNodeFromPathPT.py)

Internal.getNodesFromType: return a list of nodes of A from their type. A can be a standard node or a list of standard nodes:

```
nodes = Internal.getNodesFromType(A, 'Zone_t')
```

This function can be limited to 1, 2 or 3 levels from the starting node A by using Internal.getNodesFromType1
(See: Examples/Converter/getNodesFromTypePT.py)

Internal.getByType: return a standard node containing the matching type nodes of A. A can be a standard node or a list of standard nodes:

```
nodes = Internal.getByType(A, 'Zone_t', recursive=-1)
```

This function can be limited to 1, 2 or 3 levels from the starting node A by using recursive=1,2,3.
(See: Examples/Converter/getByTypePT.py)

Internal.getNodesFromName: return a list of nodes of a pyTree from their name (wildcards allowed):

```
nodes = Internal.getNodesFromName(A, 'cart')
```

This function can be limited to 1, 2 or 3 levels from the starting node A by using Internal.getNodesFromName
(See: Examples/Converter/getNodesFromNamePT.py)

Internal.getByName: return a standard node containing the matching name nodes of A. A can be a standard node or a list of standard nodes. Wildcards are accepted:

```
nodes = Internal.getByName(A, 'cart', recursive=-1)
```

This function can be limited to 1, 2 or 3 levels from the starting node A by using recursive=1,2,3.
(See: Examples/Converter/getByNamePT.py)

Internal.getNodeFromName: return the first node of A of given name. A must be a standard node. Wildcards are NOT accepted. This is a fast routine:

```
node = Internal.getNodeFromName(A, 'cart')
```

This function can be limited to 1, 2 or 3 levels from the starting node A by using Internal.getNodeFromName1
(See: Examples/Converter/getNodeFromNamePT.py)

Internal.getParentOfNode: return the parent node of given node. start must be a higher node in the tree. It returns p (the parent node) and c (the number of node in the parent's children list). If start is a node of a pyTree, the p[2][c] = node. If starts is a list of standard nodes, then p == start and p[c] = node. If node is not found, then p is None:

```
(p, c) = Internal.getParentOfNode(start, node)
```

(See: Examples/Converter/getParentOfNodePT.py)

Internal.getZoneDim: return the dimension of a zone node. Return ['Structured', ni, nj, nk, celldim] for structured grids, return ['Unstructured', np, ne, EltsName, celldim] for unstructured grids. np is the number of points, ne the number of elements, celldim is 0, 1, 2, 3 depndaing on element dimension, EltsName is one of NODE, BAR, TRI, QUAD, TETRA, PYRA, PENTA, NGON, MULTIPLE:

```
dims = Internal.getZoneDim(node)
```

(See: Examples/Converter/getZoneDimPT.py)

Internal.getPath: return the path of a node in A:

```
path = Internal.getPath(A, node)
```

(See: Examples/Converter/getPathPT.py)

Internal.getName: return the name of a node:

```
name = Internal.getName(node)
```

(See: Examples/Converter/getNamePT.py)

Internal.getType: return the CGNS type of a node:

```
type = Internal.getType(node)
```

(See: Examples/Converter/getTypePT.py)

Internal.getChildren: return the children of a node. Children is a node list:

```
children = Internal.getChildren(node)
```

(See: Examples/Converter/getChildrenPT.py)

Internal.getValue: return the value of a node, if the node contains a string, a float or an integer. If the node contains an array, return the numpy array:

```
val = Internal.getValue(node)
```

(See: Examples/Converter/getValueIPT.py)

1.5 Check

Internal.printTree: pretty print a pyTree or a pyTree node to screen or in a file:

```
Internal.printTree(A, file=None)
```

(See: Examples/Converter/printTreePT.py)

Internal.checkPyTree: check pyTree A following level (0: valid version node, 1: node conformity, 2: unique base name, 3: unique zone name, 4: unique BC names, 5: valid BC ranges, 6: valid opposite BC ranges for match and nearmatch, 7: referenced familyZone and familyBCs must be defined in bases, 8: valid CGNS types). Return a list of pairs of invalid nodes and error message:

```
errors = Internal.checkPyTree(A, level=-20)
```

(See: Examples/Converter/checkPyTreePT.py)

Internal.correctPyTree: correct pyTree A following level (0: valid version node, 1:node conformity, 2: unique base name, 3: unique zone name, 4: unique BC names, 5: valid BC range, 6: valid opposite BC range for match and nearMatch, 7: referenced familyZone and familyBCs must be defined in bases, 8: valid CGNS types). Non unique names are made unique. Missing families are added. Invalid nodes are removed. Return a new tree:

```
B = Internal.correctPyTree(A, level=-20)
```

(See: Examples/Converter/correctPyTreePT.py)

Internal.checkMultigrid: check if pyTree is compatible with multigrid of given level (only for structured zones). Zone dimensions, but also BCs and grid connectivities are checked. Check if coarse grids contain at least nbMinCoarseB points per direction. Check if coarse boundary and connectivity point ranges contain at least nbMinCoarseW points. Return a list of pairs of invalid nodes and error messages:

```
errors = Internal.checkMultigrid(A, level=1, nbMinCoarseB=5, nbMinCoarseW=3)
```

(See: Examples/Converter/checkMultigridPT.py)

Internal.checkSize: check if the number of points of zones in pyTree does not exceed sizeMax:

```
errors = Internal.checkSize(A, sizeMax=100000000)
```

(See: Examples/Converter/checkSizePT.py)

1.6 Copy

Internal.copyRef: copy a tree but sharing node values (in particular data numpys are shared):

```
B = Internal.copyRef(A)
```

(See: Examples/Converter/copyRefPT.py)

Internal.copyTree: fully copy a tree. Node values (in particular data numpys) are copied:

```
B = Internal.copyTree(A)
```

(See: Examples/Converter/copyTreePT.py)

1.7 Add/Remove

Internal.append: append a node by its path:

```
B = Internal.append(A, a, '/Base')
```

(See: Examples/Converter/appendPT.py)

Internal.rmNodeByPath: remove a node by its path:

```
a = Internal.rmNodeByPath(a, '/Base/cart') .or. B = Internal.rmNodeByPath(A, 'zone0')
```

(See: Examples/Converter/rmNodeByPathPT.py)

Internal.rmNodesByName: remove nodes by their name. Wildcards are accepted:

```
b = Internal.rmNodesByName(a, 'zone0') .or. B = Internal.rmNodesByName(A, 'zone0')
```

(See: Examples/Converter/rmNodesByNamePT.py)

Internal.rmNodesByType: remove nodes by their type:

```
b = Internal.rmNodesByType(a, 'Zone_t') .or. B = Internal.rmNodesByType(A, 'Zone_t')
```

(See: Examples/Converter/rmNodesByTypePT.py)

Internal.rmNodesByValue: remove nodes by their value:

```
b = Internal.rmNodesByValue(a, value) .or. B = Internal.rmNodesByType(A, value)
```

(See: Examples/Converter/rmNodesByValuePT.py)

1.8 Modify

Internal.sortByName: sort nodes by their name (alphabetical order). If recursive=True, sort also chidren nodes:

```
B = Internal.sortByName(A, recursive=True)
```

(See: Examples/Converter/sortByNamePT.py)

Internal.appendBaseName2ZoneName: add base name to zone name (resulting name is baseName_zoneName for each zone). If updateRef=True, reference to zone names in BC,... are replaced:

```
B = Internal.appendBaseName2ZoneName(A, updateRef=True)
```

(See: Examples/Converter/appendBaseName2ZoneNamePT.py)

Internal.renameNode: rename node ’name’ with ’newName’. Occurances of name elsewhere in the tree is replaced with ’newName’:

```
B = Internal.renameNode(A, name, newName)
```

(See: Examples/Converter/renameNodePT.py)

1.9 Example files

Example file : Examples/Converter/isTopTreePT.py

```
# - isTopTree (pyTree) -
import Converter.PyTree as C
import Converter.Internal as Internal

t = C.newPyTree([’Base1’, ’Base2’])
print Internal.isTopTree(t),’ is True!’
print Internal.isTopTree(t[2][1]), ’is False!’
```

Example file : Examples/Converter/isStdNodePT.py

```
# - isStdNode (pyTree) -
import Converter.PyTree as C
import Converter.Internal as Internal
import numpy

# This is a standard node
a = [’toto’, numpy.zeros(12), [], ’DataArray_t’]
print Internal.isStdNode(a)

# This is not a standard node
b = [’toto’, ’tata’]
print Internal.isStdNode(b)

# This is a list of standard nodes
c = [’titi’, numpy.zeros(13), [], ’DataArray_t’]
print Internal.isStdNode([a,c])
```

Example file : Examples/Converter/typeOfNodePT.py

```
# - typeOfNode (pyTree) -
import Converter.PyTree as C
import Converter.Internal as Internal

t = C.newPyTree([’Base1’, ’Base2’])
print Internal.typeOfNode(t)
```

Example file : Examples/Converter/isTypePT.py

```
# - isType (pyTree) -
import Converter.PyTree as C
import Converter.Internal as Internal

t = C.newPyTree([’Base1’, ’Base2’])

# This is true
print Internal.isType(t, ’CGNSTree_t’)

# This is false
print Internal.isType(t, ’CGNSBase_t’)

# Check with wildcard: answer true if the type matches the string
print Internal.isType(t, ’CGNS*’)
```

Example file : Examples/Converter/isNamePT.py

```
# - isName (pyTree) -
import Converter.PyTree as C
import Converter.Internal as Internal

t = C.newPyTree(['Base1', 'Base2'])

# This is false
print Internal.isName(t[2][1], 'Base3')

# This is true if node name matches the string
print Internal.isName(t[2][1], 'Base*')
```

Example file : Examples/Converter/isValuePT.py

```
# - isValue (pyTree) -
import Converter.PyTree as C
import Converter.Internal as Internal
import numpy

# Check a scalar value
node = ['node1', 1., [], 'DataArray_t']
print Internal.isValue(node, 1.)

# Check a numpy array value
node = ['node1', numpy.zeros(10), [], 'DataArray_t']
print Internal.isValue(node, numpy.zeros(10))

# Check a string value
node = ['node1', 'toto', [], 'DataArray_t']
print Internal.isValue(node, 'toto')
```

Example file : Examples/Converter/setValueIPT.py

```
# - setValue (pyTree) -
import Converter.PyTree as C
import Converter.Internal as Internal
import numpy

node = ['node1', 12., [], 'DataArray_t']

# Set a scalar value in node
Internal.setValue(node, 1.) ; print node

# Set a numpy array in node
Internal.setValue(node, numpy.zeros(10)) ; print node

# Set an array as a list
Internal.setValue(node, [1.,12.,13.]) ; print node
```

Example file : Examples/Converter/setNamePT.py

```
# - setName (pyTree) -
import Converter.PyTree as C
import Converter.Internal as Internal

node = ['node1', 1., [], 'DataArray_t']
Internal.setName(node, 'myNode') ; print node
```

Example file : Examples/Converter/setTypePT.py

```
# - setType (pyTree) -
import Converter.PyTree as C
import Converter.Internal as Internal

node = ['node1', 1., [], 'DataArray_t']
Internal.setType(node, 'Zone_t') ; print node
```

Example file : Examples/Converter/createNodePT.py

```
# - createNode (pyTree) -
import Converter.PyTree as C
import Converter.Internal as Internal

# Create a node named myNode, of type DataArray_t and with value 1.
node = Internal.createNode('myNode', 'DataArray_t', value=1., children=[])
print node

# Create a node named myNode, of type DataArray_t and with an array value
node = Internal.createNode('myNode', 'DataArray_t', value=[12.,14.,15.],
                           children=[])
print node
```

Example file : Examples/Converter/addChildPT.py

```
# - addChild (pyTree) -
import Converter.PyTree as C
import Converter.Internal as Internal

node = Internal.createNode('myNode', 'DataArray_t', value=1., children=[])
child1 = Internal.createNode('child1', 'DataArray_t', value=2., children=[])
child2 = Internal.createNode('child2', 'DataArray_t', value=3., children=[])

# add children nodes to node
Internal.addChild(node, child1, pos=-1) # at the end
Internal.addChild(node, child2, pos=0) # first
print node
```

Example file : Examples/Converter/createChildPT.py

```
# - createChild (pyTree) -
import Converter.PyTree as C
import Converter.Internal as Internal

node = Internal.createNode('myNode', 'DataArray_t', value=1., children=[])
Internal.createChild(node, 'childName', 'DataArray_t', value=2., children=[])
print node
```

Example file : Examples/Converter/createUniqueChildPT.py

```
# - createUniqueChild (pyTree) -
import Converter.PyTree as C
import Converter.Internal as Internal

node = Internal.createNode('myNode', 'DataArray_t', value=1., children=[])
Internal.createUniqueChild(node, 'childName', 'DataArray_t', value=2.)
Internal.createUniqueChild(node, 'childName', 'DataArray_t', value=2.)
print node
```

Example file : Examples/Converter/getNodeFromPathPT.py

```
# - getNodeFromPath (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Converter.Internal as Internal

a = G.cart( (0,0,0), (1,1,1), (10,10,10) )
t = C.newPyTree(['Base']) ; t[2][1][2].append(a)

# Return GridCoordinates node
coords = Internal.getNodeFromPath(t, 'Base/cart/GridCoordinates')
print coords

# Return GridCoordinates node (path is relative to input node)
coords = Internal.getNodeFromPath(a, 'GridCoordinates')
print coords
```

Example file : Examples/Converter/getNodesFromTypePT.py

```
# - getNodesFromType (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Converter.Internal as Internal

a = G.cart( (0,0,0), (1,1,1), (10,10,10) )
t = C.newPyTree(['Base']) ; t[2][1][2].append(a)

# Return nodes of type 'Zone_t'
zones = Internal.getNodesFromType(t, 'Zone_t') ; print zones

# Limit search to first level children
zones = Internal.getNodesFromType1(t, 'Zone_t') ; print zones
```

Example file : Examples/Converter/getByTypePT.py

```
# - getByType (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Converter.Internal as Internal

a = G.cart( (0,0,0), (1,1,1), (10,10,10) )
t = C.newPyTree(['Base']) ; t[2][1][2].append(a)

# Return a standard node containing nodes of type 'Zone_t' as children
zones = Internal.getByType(t, 'Zone_t') ; print zones
```

Example file : Examples/Converter/getNodesFromNamePT.py

```
# - getNodesFromName (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Converter.Internal as Internal

a = G.cart( (0,0,0), (1,1,1), (10,10,10) )
t = C.newPyTree(['Base']) ; t[2][1][2].append(a)

# Return nodes named 'cart'
nodes = Internal.getNodesFromName(t, 'cart') ; print nodes

# Return the 3 coordinate nodes
nodes = Internal.getNodesFromName(t, 'Coordinate*') ; print nodes
```

Example file : Examples/Converter/getByNamePT.py

```
# - getByName (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Converter.Internal as Internal

a = G.cart( (0,0,0), (1,1,1), (10,10,10) )
t = C.newPyTree(['Base']) ; t[2][1][2].append(a)

# Return a standard node containing nodes of name 'cart' as children
nodes = Internal.getByName(t, 'cart') ; print nodes
```

Example file : Examples/Converter/getNodeFromNamePT.py

```
# - getNodeFromName (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Converter.Internal as Internal

a = G.cart( (0,0,0), (1,1,1), (10,10,10) )
```

```
t = C.newPyTree(['Base']) ; t[2][1][2].append(a)

# Return the node named 'cart'
node = Internal.getNodeFromName(t, 'cart') ; print node
```

Example file : Examples/Converter/getParentOfNodePT.py

```
# - getParentOfNode (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Converter.Internal as Internal

a = G.cart( (0,0,0), (1,1,1), (10,10,10) )
t = C.newPyTree(['Base']) ; t[2][1][2].append(a)

(p, c) = Internal.getParentOfNode(t, a) ; print p
```

Example file : Examples/Converter/getZoneDimPT.py

```
# - getZoneDim (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Converter.Internal as Internal

a = G.cart( (0,0,0), (1,1,1), (10,10,10) )
dim = Internal.getZoneDim(a) ; print dim

a = G.cartTetra( (0,0,0), (1,1,1), (10,10,10) )
dim = Internal.getZoneDim(a) ; print dim
```

Example file : Examples/Converter/getPathPT.py

```
# - getPath (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Converter.Internal as Internal

a = G.cart( (0,0,0), (1,1,1), (10,10,10) )
t = C.newPyTree(['Base']) ; t[2][1][2] += [a]
print Internal.getPath(t, a)
```

Example file : Examples/Converter/getNamePT.py

```
# - getName (pyTree) -
import Converter.Internal as Internal
import Generator.PyTree as G

a = G.cart( (0,0,0), (1,1,1), (10,10,10) )
print Internal.getName(a)
```

Example file : Examples/Converter/getTypePT.py

```
# - getType (pyTree) -
import Converter.Internal as Internal
import Generator.PyTree as G

a = G.cart( (0,0,0), (1,1,1), (10,10,10) )
print Internal.getType(a)
```

Example file : Examples/Converter/getChildrenPT.py

```
# - getChildren (pyTree) -
import Converter.Internal as Internal
import Generator.PyTree as G

a = G.cart( (0,0,0), (1,1,1), (10,10,10) )
children = Internal.getChildren(a)
print len(children)
```

Example file : Examples/Converter/getValueIPT.py

```
# - getValue of a node (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Converter.Internal as Internal

# Structured array
Ni = 40; Nj = 50; Nk = 20
a = G.cart((0,0,0), (1./(Ni-1), 0.5/(Nj-1),1./(Nk-1)), (Ni,Nj,Nk))

# Get value of a zone node
print Internal.getValue(a)

# Get type of a zone
node = Internal.getNodeFromName(a, 'ZoneType')
print Internal.getValue(node)
```

Example file : Examples/Converter/printTreePT.py

```
# - printTree (pyTree) -
import Converter.PyTree as C
import Converter.Internal as Internal
import Generator.PyTree as G

a = G.cart( (0,0,0), (1,1,1), (10,10,10) )
t = C.newPyTree(['Base']) ; t[2][1][2] += [a]
Internal.printTree(t)
```

Example file : Examples/Converter/checkPyTreePT.py

```
# - checkPyTree (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Connector.PyTree as X
import Converter.Internal as Internal

a = G.cart( (0,0,0), (1,1,1), (10,10,10) )
b = G.cart( (9,0,0), (1,1,1), (10,10,10) )
a = C.addBC2Zone(a, 'wall1', 'BCWall', 'imin')
t = C.newPyTree(['Base']) ; t[2][1][2] += [a,b]
t = X.connectMatch(t)

# check nodes conformity
errors = Internal.checkPyTree(t, level=1)
# check unique base names
errors += Internal.checkPyTree(t, level=2)
# check unique zone names
errors += Internal.checkPyTree(t, level=3)
# check unique BC names
errors += Internal.checkPyTree(t, level=4)
# check BC ranges
errors += Internal.checkPyTree(t, level=5)
# check opposite ranges
errors += Internal.checkPyTree(t, level=6)
# check family definition
errors += Internal.checkPyTree(t, level=7)
print errors
```

Example file : Examples/Converter/correctPyTreePT.py

```
# - correctPyTree (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Connector.PyTree as X
import Converter.Internal as Internal
```

```
a = G.cart( (0,0,0), (1,1,1), (10,10,10) )
b = G.cart( (9,0,0), (1,1,1), (10,10,10) )
a = C.addBC2Zone(a, 'wall1', 'BCWall', 'imin')
t = C.newPyTree(['Base']) ; t[2][1][2] += [a,b]
t = X.connectMatch(t)

t = Internal.correctPyTree(t)
```

Example file : Examples/Converter/checkMultigridPT.py

```
# - checkMultigrid (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Connector.PyTree as X
import Converter.Internal as Internal

a = G.cart( (0,0,0), (1,1,1), (11,11,11) )
b = G.cart( (10,0,0), (1,1,1), (11,11,11) )
a = C.addBC2Zone(a, 'wall1', 'BCWall', 'imin')
t = C.newPyTree(['Base']) ; t[2][1][2] += [a,b]
t = X.connectMatch(t)
C.convertPyTree2File(t, 'out.cgns')

# check multigrid compatibility
errors = Internal.checkMultigrid(t, level=1)
print errors
```

Example file : Examples/Converter/checkSizePT.py

```
# - checkSize (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Connector.PyTree as X
import Converter.Internal as Internal

a = G.cart( (0,0,0), (1,1,1), (11,11,11) )
b = G.cart( (10,0,0), (1,1,1), (11,11,11) )
a = C.addBC2Zone(a, 'wall1', 'BCWall', 'imin')
t = C.newPyTree(['Base']) ; t[2][1][2] += [a,b]
t = X.connectMatch(t)
C.convertPyTree2File(t, 'out.cgns')

# check nodes conformity
errors = Internal.checkSize(t, sizeMax=500)
print errors
```

Example file : Examples/Converter/copyRefPT.py

```
# - copyRef (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cart((0,0,0), (1,1,1), (10,10,10))
t = C.newPyTree(['Base']) ; t[2][1][2] += [a]

# t2 and t are sharing values (also data numpys)
t2 = C.copyRef(t)
```

Example file : Examples/Converter/copyTreePT.py

```
# - copyTree (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G

a = G.cart((0,0,0), (1,1,1), (10,10,10))
```

```
t = C.newPyTree(['Base']) ; t[2][1][2] += [a]

# t2 is a full copy of t (values/numpys are also copied)
t2 = C.copyTree(t)
```

Example file : Examples/Converter/appendPT.py

```
# - append (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Converter.Internal as Internal

t = C.newPyTree(['Base', 'Base2'])

a = G.cart( (0,0,0), (1,1,1), (10,10,10) )
t = Internal.append(t, a, '/Base')
C.convertPyTree2File(t, 'out.cgns')
```

Example file : Examples/Converter/rmNodeByPathPT.py

```
# - rmNodeByPath (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Converter.Internal as Internal

t = C.newPyTree(['Base', 'Base2'])
for i in xrange(10):
    a = G.cart( (0,0,0), (1,1,1), (10,10,10) )
    a[0] = 'Cart'+str(i)
    t[2][1][2].append(a)

t = Internal.rmNodeByPath(t, '/Base/Cart2')
C.convertPyTree2File(t, 'out.cgns')
```

Example file : Examples/Converter/rmNodesByNamePT.py

```
# - rmNodesByName (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Converter.Internal as Internal

t = C.newPyTree(['Base', 'Base2'])
for i in xrange(10):
    a = G.cart( (0,0,0), (1,1,1), (10,10,10) )
    a[0] = 'Cart'+str(i)
    t[2][1][2].append(a)

t = Internal.rmNodesByName(t, 'Cart0')
t = Internal.rmNodesByName(t, 'Cart*')
C.convertPyTree2File(t, 'out.cgns')
```

Example file : Examples/Converter/rmNodesByTypePT.py

```
# - rmNodesByType (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Converter.Internal as Internal

t = C.newPyTree(['Base', 'Base2'])
for i in xrange(10):
    a = G.cart( (0,0,0), (1,1,1), (10,10,10) )
    a[0] = 'Cart'+str(i)
    t[2][1][2].append(a)

t = Internal.rmNodesByType(t, 'Zone_t')
C.convertPyTree2File(t, 'out.cgns')
```

Example file : Examples/Converter/rmNodesByValuePT.py

```
# - rmNodesByValue (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Converter.Internal as Internal

a = G.cart( (0,0,0), (1,1,1), (10,10,10) )
a = C.fillEmptyBCWith(a, 'far', 'BCFarfield')

a = Internal.rmNodesByValue(a, 'BCFarfield')
t = C.newPyTree(['Base']) ; t[2][1][2] += [a]
C.convertPyTree2File(t, 'out.cgns')
```

Example file : Examples/Converter/sortByNamePT.py

```
# - sortByName (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Converter.Internal as Internal

t = C.newPyTree(['Base'])

a = G.cart( (0,0,0), (1,1,1), (10,10,10) ) ; a[0] = 'b'
b = G.cart( (12,0,0), (1,1,1), (10,10,10) ) ; b[0] = 'a'
t[2][1][2] += [a,b]
t = Internal.sortByName(t)
C.convertPyTree2File(t, 'out.cgns')
```

Example file : Examples/Converter/appendBaseName2ZoneNamePT.py

```
# - appendBaseName2ZoneName (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Converter.Internal as Internal

t = C.newPyTree(['Base'])

a = G.cart( (0,0,0), (1,1,1), (10,10,10) ) ; a[0] = 'b'
b = G.cart( (11,0,0), (1,1,1), (10,10,10) ) ; b[0] = 'a'
t[2][1][2] += [a,b]
t = Internal.appendBaseName2ZoneName(t)
C.convertPyTree2File(t, 'out.cgns')
```

Example file : Examples/Converter/renameNodePT.py

```
# - renameName (pyTree) -
import Converter.PyTree as C
import Generator.PyTree as G
import Converter.Internal as Internal
import Connector.PyTree as X

t = C.newPyTree(['Base'])

a = G.cart( (0,0,0), (1,1,1), (10,10,10) )
b = G.cart( (9,0,0), (1,1,1), (10,10,10) )
t[2][1][2] += [a,b]
t = X.connectMatch(t)

# Change the zone named 'cart' and its reference in BCMatch (GridConnectivity)
t = Internal.renameNode(t, 'cart', 'myCart')
C.convertPyTree2File(t, 'out.cgns')
```